State-of-the-art in Smith-Waterman Protein Database Search on HPC Platforms

Enzo Rucci, Carlos García, Guillermo Botella, Armando De Giusti, Marcelo Naiouf and Manuel Prieto-Matías

Abstract Searching biological sequence database is a common and repeated task in bioinformatics and molecular biology. The Smith-Waterman algorithm is the most accurate method for this kind of search. Unfortunately, this algorithm is computationally demanding and the situation gets worse due to the exponential growth of biological data in the last years. For that reason, the scientific community has made great efforts to accelerate Smith-Waterman biological database searches in a wide variety of hardware platforms. We give a survey of the state-of-the-art in Smith-Waterman protein database search, focusing on four hardware architectures: central processing units, graphics processing units, field programmable gate arrays and Xeon Phi coprocessors. After briefly describing each hardware platform, we

Enzo Rucci
Instituto de Investigación en Informática LIDI (III-LIDI), Universidad Nacional de La Plata, Buenos Aires, Argentina, e-mail: erucci@lidi.info.unlp.edu.ar

Carlos García
Dpto. Arquitectura de Computadores y Automática Universidad Complutense de Madrid, Madrid, Spain e-mail: garsanca@ucm.es

Guillermo Botella
Dpto. Arquitectura de Computadores y Automática Universidad Complutense de Madrid, Madrid, Spain e-mail: gbotella@fdi.ucm.es

Armando De Giusti
Instituto de Investigación en Informática LIDI (III-LIDI), Universidad Nacional de La Plata, Buenos Aires, Argentina, e-mail: degiusti@lidi.info.unlp.edu.ar

Marcelo Naiouf
Instituto de Investigación en Informática LIDI (III-LIDI), Universidad Nacional de La Plata, Buenos Aires, Argentina, e-mail: mnaiouf@lidi.info.unlp.edu.ar

Manuel Prieto-Matías
Dpto. Arquitectura de Computadores y Automática Universidad Complutense de Madrid, Madrid, Spain e-mail: mpmatias@ucm.es

The final authenticated version is available online at https://doi.org/10.1007/978-3-319-41279-5_6

analyse temporal evolution, contributions, limitations and experimental work and the results of each implementation. Additionally, as energy efficiency is becoming more important every day, we also survey performance/power consumption works. Finally, we give our view on the future of Smith-Waterman protein searches considering next generations of hardware architectures and its upcoming technologies.

1 Introduction

Searching biological sequence database is a common and repeated task in bioinformatics and molecular biology. In a typical search operation, biological sequences with unknown functionalities (usually referred as query sequences) are aligned to a database of known sequences to find similarities. The alignment process computes a score that represents the degree of similarity between each pair of query and database sequences. Sequence alignment methods are classified as either global or local. Global alignments try to maximize the number of matches between the two sequences along their entire lengths and are useful when the sequences are similar. On the other hand, local alignments try to maximize the number of matches between small portions of the two sequences. This kind of alignment exposes much better similarity between unrelated sequences and, at the same time, leads to more biologically relevant results [19]. The Smith-Waterman (SW) algorithm is the most accurate method for local sequence alignment. This algorithm is based on dynamic programming approach and its high sensitivity comes from exploring all the possible alignments between two sequences. Unfortunately, this method is computationally demanding and the situation gets worse due to the exponential growth of biological data in the last years. One frequently used approach to speed up this time demanding operation is to introduce heuristics in order to reduce the search space. Heuristics usually produce considerably good results. However, they are deficient in searching the best match subsequences and, in consequence, are not guaranteed to discover the optimal alignment. For that reason, the scientific community has made great efforts to accelerate SW protein database searches through High-Performance Computing (HPC) in a wide variety of hardware platforms. This chapter gives a survey of the state-of-the-art in SW protein database search, focusing on four hardware architectures: Central Processing Units (CPU), Graphics Processing Units (GPU), Field Programmable Gate Arrays (FPGA) and Xeon Phi coprocessors. After briefly describing each hardware platform, we analyse temporal evolution, contributions, limitations and experimental work and the results of each implementation. Additionally, as energy efficiency is becoming more important every day, we also survey performance/power consumption works. Finally, we give our view on the future of SW protein searches considering next generations of hardware architectures and its upcoming technologies.

The rest of the chapter is organised as follows: Section 2 briefly describes the considered hardware platforms. Section 3 introduces the basic concepts of the SW

algorithm. Section 4 reviews hardware acceleration of SW protein database search. Section 5 overviews performance-power consumption evaluations on SW context. Section 6 gives our view on the future of SW protein searches considering next generations of hardware architectures. Finally, Section 7 presents the conclusions of this chapter.

2 Hardware Platforms

Scientific community has made great efforts to accelerate Smith-Waterman biological database searches in a wide variety of hardware architectures. Next there is a brief description for each hardware platform considered in this work.

2.1 CPU

Traditional chip design was guided by increasing transistor count and clock speed, which enabled designers to implement many advanced techniques that permitted to increment Instruction Level Parallelism (ILP) and, in consequence, led to improved application performance. However, at the beginning of this century, this design process got stuck due to two reasons:

- extracting more ILP from programmes became a hard task and
- increasing clock frequency reached unsustainable power consumption and heat generation levels.

Multi-core processors arose as a solution to this problem. Hardware vendors decided to integrate two or more computational cores within the same chip. Even though these cores are simpler and slower, when combined, they permit enhancing the global performance of the processor while making an efficient use of energy [29]. Its introduction also affected application programmers because explicit parallelism should be exploited to take advantage of multi-core hardware; in particular, both data and task parallelism. The first multi-core CPUs were simply two processors on the same die but later generations incorporated more cores, additional cache levels and better interconnection networks, among other features.

Currently, the main CPU vendor is Intel followed by AMD. In 2015, Intel presented the Skylake micro-architecture introducing the first processors of this family and more models were announced to the next two years. In particular, the high-performance Xeon line will incorporate several improvements, such as support for more sockets, channels of DDR4 memory and PCIe slots. Additionally, these processors will include AVX-512 vectorial instruction set [1](a 512-bit extension of

[1] AVX-512 Extensions: `https://software.intel.com/en-us/blogs/additional-avx-512-instructions`

the current Advanced Vector Extensions with 256-bit width) and will give support to integrated FPGAs [37]. According to Intel, the next two micro-architectures will be available in 2016 (Kaby Lake) and 2017 (Cannonlake). Kaby Lake will be an upgraded version of Skylake while Cannonlake will shrink the fabrication technology to 10nm [15].

AMD introduced three different micro-architectural families in 2011. The Fusion family corresponds to the Accelerated Processing Units that integrate CPUs and GPUs on the same chip. On the other hand, the Bobcat family was designed for low-power and low-cost devices while the Bulldozer family is oriented to desktop computers and servers. Finally, the next AMD micro-architecture is named Zen and will be available by the end of 2016. Zen's main purpose consists in improving performance per core more than increasing number of cores or hardware threads. In particular, some preliminary reports state up to 40% more instructions per clock cycle [31]. Unlike previous micro-architectural families, Zen will adopt simultaneous multi-threading capabilities and will be developed using 14nm fabrication technology.

2.2 GPU

GPUs were originally developed for computer games and its designs were orientated for that purpose. The first non-graphic applications were programmed adapting primitives from graphic languages like OpenGL or DirectX. In the last decade, GPU architectures were modified and several programming libraries were introduced that permitted avoiding graphic primitives. These changes increased GPU usage in significant manner. Nowadays, they have consolidated as general-purpose accelerators in HPC community due to the increasing compute power and energy efficiency.

Currently, most popular programming languages for GPUs are CUDA [34], OpenCL [46] and, in lesser extent, OpenACC [36]. While these languages reduces programming cost compared to initial graphic languages, they still represent a hard task because they significantly differ from traditional CPU's programming model. Therefore, programmers must learn specific GPU knowledge to achieve high-performance applications. For example, common optimisation techniques comprise increasing hardware occupancy, exploiting memory hierarchy, organising memory accesses, avoiding divergent branches, among others.

The two main GPU vendors are NVIDIA and AMD. In 2011, AMD introduced Graphics Core Next (GCN) architecture, which is the basis for its individual and integrated GPUs. GCN was designed to achieve high performance not only in graphic applications but also in general purpose tasks [43]. One of the main AMD innovations in the last years is High-Bandwidth Memory (HBM) technology, a new type of *3D memory* that can be used in CPUs and GPUs. This kind of memory has several advantages: significant space savings and increased bandwidth and energetic efficiency [3]. HBM has been incorporated in AMD GPUs codenamed Fiji from

GCN architecture, introduced in 2015, and more AMD cards with this technology will be available in 2016.

Current NVIDIA GPUs are based on Maxwell architecture, which was presented in 2014. Maxwell family redesigned Streaming Multiprocessor architecture, the heart of each NVIDIA GPU, and also the memory hierarchy [13]. These changes allowed Maxwell to improve performance and power efficiency in relation to its predecessor Kepler. NVIDIA has announced its next architecture codenamed Pascal for 2016, which will include HBM adoption and 16nm manufacturing process. According to NVIDIA, Pascal will improve performance, performance per watt, memory capacity and bandwidth of Maxwell [32].

2.3 FPGA

FPGAs are reconfigurable integrated circuits comprising programmable interconnections that join programmable logic blocks, embedded memory blocks and digital signal processor blocks. Communication to the outside is performed through I/O blocks, which are arranged in a ring form around the circumference of these devices. As opposed to CPUs and GPUs, FPGA resources may be configured and linked together to create custom instruction pipelines through which data is processed. Also, they work at lower clock frequencies and have lower peak performances. However, since FPGAs can configure its hardware for each specific application, they usually reach better performance efficiencies. Additionally, they are normally more efficient from energetic point of view as there is no silicon waste [42, 49].

Since its development, FPGAs have significantly evolved continuously incrementing its available resources and incorporating features like standards for interconnection networks and high-speed I/O. At the beginning, FPGAs were used for digital signal processing. However, in the last few years, there are two clear trends to enlarge FPGA usage in other application domains. The first comprises the increasing integration of FPGAs with CPUs due to accelerators consolidation in HPC community as a way of improving performance while keeping power efficiency. In particular, the two main FPGA makers Xilinx and Altera have established different agreements with important CPU vendors to develop hybrid CPU-FPGA architectures. IBM has announced a strategic partnership with Xilinx to enable higher performance and energy-efficient applications through FPGA-enabled workload acceleration on IBM POWER-based systems [16]. On its behalf, Altera has been recently acquired by Intel and they plan to combine Altera's FPGA products with Intel Xeon processors as highly customized, integrated products [17]. The second trend consists in reducing FPGA programming cost. Generally, digital design verification and creation have involved the use of Hardware Description Languages (HDLs), like Verilog and VHDL. However, HDLs are tedious, error prone and affected by an extra abstraction layer as they contain the additional concept of time. Currently, both Altera and Xilinx are working on high-level tools

that seek to reduce the programming cost of these devices; in particular, through OpenCL standard [2, 53].

2.4 Xeon Phi

The Xeon Phi is a recent many-core coprocessor developed by Intel for HPC applications. In its current generation, the Xeon Phi features up to 61 x86 pentium cores with extended vector processing units (512-bit) named Knight Corner (KNC) and simultaneous multi-threading capabilities (four hardware threads per core). Each core integrates an L1 cache and has an associated fully coherent L2 cache. Additionally, a high-speed ring interconnection allows data transfer among all the L2 caches and the memory subsystem.

The Xeon Phi offers two execution modes: *offload* and *native*. In the offload mode, the Xeon Phi acts as a coprocessor. It takes on computationally demanding parts of programmes delegated by the CPU. In the native mode, the Xeon Phi runs as a completely standalone computing system. In this mode, applications can use solely the resources of the coprocessor.

From a programming point of view, one of the main advantages of this platform is the support of existing parallel programming models traditionally used on HPC systems such as the OpenMP or MPI paradigms, which simplifies code development and improves portability over other alternatives based on accelerator-specific programming languages such as CUDA or OpenCL.

With regard to the future of Intel many-core coprocessors, Intel has announced the next generation, called Knights Landing, which is planned to run HPC systems in 2016. Among the main differences posted, the chip will be built with 14nm technology and be able to operate as a standalone CPU rather than as a coprocessor. It will also incorporate Intel Silvermont processors with AVX-512 vector capabilities, unifying in this way vector extensions with general purpose Intel Xeon Skylake processors. Lastly, main memory will have a stacked organisation, similar to HBM proposal [37].

3 Smith-Waterman Algorithm

In 1970, Saul Needleman and Christian Wunsch introduced an algorithm to compute optimal global alignment between two biological sequences, known as the Needleman-Wunsch algorithm [33]. Later, in 1981, Temple Smith and Michael Waterman proposed a variant of the Needleman-Wunsch algorithm to find the optimal local alignment of two sequences [44]. The SW method has been used as the basis for many subsequent algorithms and is often employed as a benchmark when comparing different alignment techniques [14]. Its strength comes from

the guarantee of discovering optimal alignment because it explores all possible alignments between the pair of sequences.

The SW algorithm computes the optimal local alignment between two sequences following a dynamic programming approach and can be divided in two stages: (1) similarity matrix (also called alignment matrix) filling, to obtain optimal alignment score; and (2) traceback, to obtain optimal alignment.

1. *Similarity matrix filling:* given two sequences $q = q_1q_2q_3 \dots q_m$ and $d = d_1d_2d_3 \dots d_n$, SW fills a matrix H which keeps track of the degree of similarity between them. The recurrence relations for the SW algorithm with the modifications of Gotoh [12] for handling multiple sized gap penalties are shown below:

$$H_{i,j} = max\{0\ ,\ \ H_{i-1,j-1} + SM(q_i, d_j),\ \ E_{i,j},\ \ F_{i,j}\} \tag{1}$$

$$E_{i,j} = max\{H_{\,i,j-1} - G_{oe},\ \ E_{i,j-1} - G_e\} \tag{2}$$

$$F_{i,j} = max\{H_{\,i-1,j} - G_{oe},\ \ F_{i-1,j} - G_e\} \tag{3}$$

   The residues of sequence q, usually called *query sequence*, label the rows. In similar way, the residues of sequence d, usually called *database sequence*, label the columns. $H_{i,j}$ represents the score for aligning the prefixes of q and d ending at position i and j, respectively. $E_{i,j}$ and $F_{i,j}$ are the scores ending with a gap involving the first i residues of q and the first j residues of d, respectively. SM is the *substitution matrix* which defines the substitution scores for all residue pairs. Generally SM rewards with a positive value when q_i and d_j are identical or relatives, and punishes with a negative value otherwise. Common substitution matrices for protein alignment are BLOSUM or PAM families. G_{oe} is the sum of gap open and gap extension penalties while G_e is the gap extension penalty. The recurrences should be calculated with $1 \leq i \leq m$ and $1 \leq j \leq n$, after initializing H, E and F with 0 when $i = 0$ or $j = 0$. The maximal alignment score in the matrix H is the optimal local alignment score S.

2. *Traceback:* Based on the position in matrix H where the value S was found, a traceback procedure is performed to obtain the pair of segments with maximum similarity, until a position whose value is zero is reached (this being the starting alignment point of the segments). These two segments represent the best local alignment.

The SW algorithm has quadratic time complexity. To compute optimal alignments, this method has quadratic spatial complexity. However, computing optimal alignment scores do not require storing full similarity matrix and can be computed in linear space complexity.

Figure 1 shows the calculation of four cells ($H_{1,1}$, $H_{1,2}$, $H_{2,1}$ and $H_{2,2}$) in the similarity matrix corresponding to the SW alignment between protein sequences CAWHEAET (q) and CITAGWHEE (d). BLOSUM62 was selected as the scoring

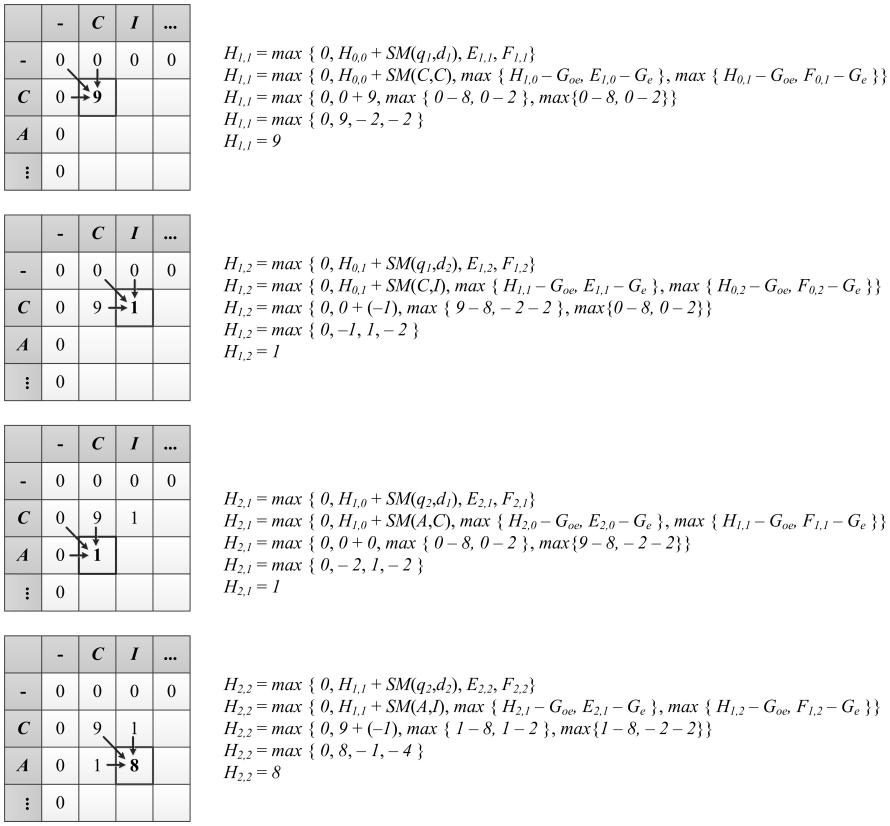

Fig. 1 Calculation of four cells in the similarity matrix corresponding to the SW alignment between protein sequences CAWHEAET and CITAGWHEE

matrix, and gap insertion and extension penalties were set to 6 and 2, respectively. After initializing H, E and F with zero when $i = 0$ or $j = 0$, the other cells in the similarity matrix are computed according to Eq. 1. For example, the cell $H_{1,1}$ in Fig. 1 is 9 because that is the maximum of 0, 9 (the upper-left neighbour plus the similarity score from BLOSUM62 substitution matrix, $0 + 9 = 9$), -2 (the alignment score ending with a gap in the query sequence, $max\{0-8, 0-2\} = -2$) and -2 (the alignment score ending with a gap in the database sequence, $max\{0-8, 0-2\} = -2$).

It is important to note that no cell value can be less than zero. Additionally, there is a strict order of computation in matrix H due to the data dependences inherent to this problem. Any cell in matrix H has a dependence on three cells: the one to the left, the one above and the one from the upper left diagonal. This dependence is illustrated in Fig. 1 through arrows.

Once all values in matrix H were computed, a traceback procedure is performed to obtain the best local alignment. Figure 2 illustrates the complete similarity matrix

	-	C	I	T	A	G	W	H	E	E
-	0	0	0	0	0	0	0	0	0	0
C	0	9	1	0	0	0	0	0	0	0
A	0	1	8	1	**4**	0	0	0	0	0
W	0	0	0	6	0	**15**	7	5	3	1
H	0	0	0	0	4	7	**13**	**15**	7	5
E	0	0	0	0	0	5	5	13	**20**	12
A	0	0	0	0	4	3	5	5	12	19
E	0	0	0	0	0	1	1	5	10	17
T	0	0	0	5	0	0	0	1	8	9

Best local alignment

A W H — E
A G W H E

Fig. 2 Illustration of Smith-Waterman alignment between protein sequences CAWHEAET and CITAGWHEE

corresponding to the SW alignment shown in Fig. 1. The traceback starts in *S* position and then works backwards (upwards and to the left). Moving upwards inserts a gap in the database sequence while moving to the left inserts a gap in the query sequence. The traceback procedure terminates when a position whose value is zero is reached (this being the starting alignment point of the segments). These two segments represent the best local alignment. In this example, the optimal alignment score is 20 while the best local alignment is shown under the similarity matrix.

4 Acceleration of SW Protein Database Search

As explained in Section 1, dynamic programming algorithms can be too computationally expensive to be used in biological database searches. In fact, due to the exponential growth in biological data, heuristic approaches are not enough in some occasions [19]. This is where parallelism exploitation becomes fundamental to accelerate this kind of searches. This section focuses on the state-of-the-art of SW algorithm acceleration. First of all, we study the data dependences of SW algorithm and the possible ways of parallelize it. Next, we describe the available implementations for four different hardware platforms: CPU, GPU, FPGA and Xeon Phi.

4.1 Data dependences and parallelism

The most computational expensive part of SW algorithm is the similarity matrix filling. Even though the data dependences described in Section 3 restrict the ways in

that the similarity matrix can be computed, the SW computation has some inherent parallelism that can be exploited to reduce its computational cost. In general, accelerated implementations adopt one of the following two approaches:

- In the intra-task parallelism approach, the parallelism within a single pair of sequences is exploited. The implementations following this approach compute several anti-diagonal cells in parallel, since these computations are independent among them. It is also possible to compute several cells in a row or a column at the same time; however, a subsequent adjustment mechanism is required to maintain algorithm coherency due to data dependence ignorance.
- Inter-task parallelism is based on performing several pairwise alignments concurrently. Its backbone is based on null data-dependence between alignments, which turns the problem into an embarrassingly parallel one.

Both approaches have been extensively explored by scientific community in a wide variety of hardware platforms. In the following subsection, we describe the works based on CPU, GPU, FPGA and Xeon Phi.

4.2 Available Implementations

Cell updates per second (CUPS) is a commonly used performance measure in the Smith-Waterman context, because it allows removal of the dependence on the query sequences and the databases utilized for the different tests as well as the hardware device. A CUPS represents the time for a complete computation of one cell in matrix H, including all memory operations and the corresponding computation of the values in the E and F arrays. Given a query sequence Q and a database D, the GCUPS (billion cell updates per second) value is calculated by:

$$\frac{|Q| \times |D|}{t \times 10^9} \tag{4}$$

where $|Q|$ is the total number of residues in the query sequence, $|D|$ is the total number of residues in the database and t is the runtime in seconds [24].

Next, we describe the most notable implementations according to the hardware architecture employed.

4.2.1 CPU Implementations

The first efforts to accelerate SW algorithm date back to the 90s. Alpern *et al.* [1] proposed several techniques including a parallel implementation that used microparallelism by dividing the 64-bit Z-buffer registers of the Intel Paragon i860 processor into four parts. This approach allowed comparing the query sequence and four database sequences at the same time. As a result, they reached a 5× speed-up compared to a conventional implementation.

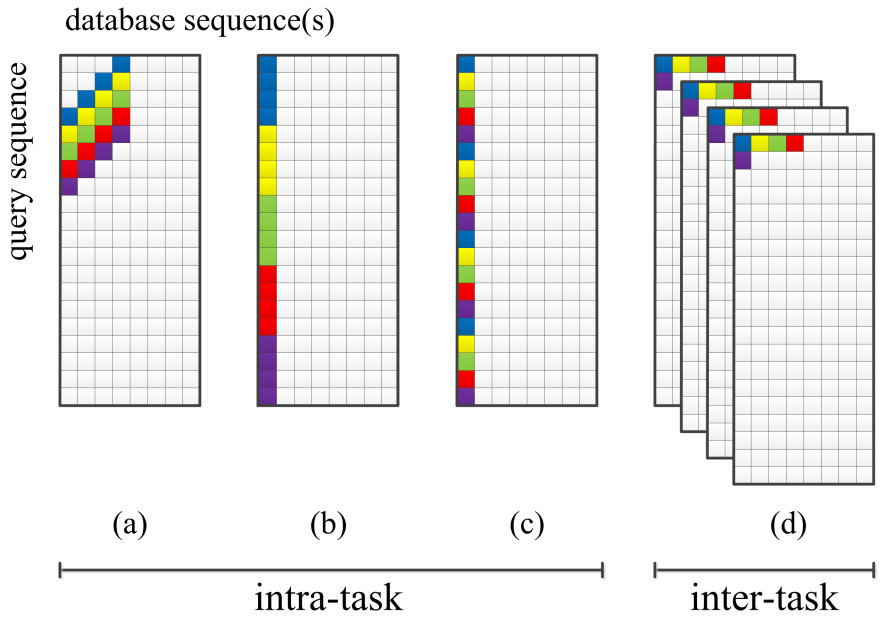

Fig. 3 Approaches to vectorisation in similarity matrix computations (adapted from [38]): (a) Vectorisation along the anti-diagonals, proposed by Wozniak [52]; (b) Vectorisation along the query sequence, proposed by Rognes and Seeberg [39]; (c) Vectorisation along the query sequence (striped approach), proposed by Farrar [9]; (d) Vectorisation along multiple database sequences, proposed by Alpern *et al.* [1].

The intra-register parallelism of the previous approach would be simpler and easier with the future introduction of small vectorial capabilities from processor vendors. With the rise of multimedia applications, general purpose processors incorporated SIMD technology, like the MMX, SSE, AVX or AltiVec extensions [38]. In general, four different approaches can be identified to vectorisation in similarity matrix computations. Figure 3 illustrates these approaches.

Most efforts focused on intra-task parallelism. In 1997, Wozniak [52] presented a parallel implementation for a Sun Ultra Sparc processor that exploited the SIMD video instructions to compute several anti-diagonal cells in parallel, since they have no dependences among them. Unfortunately, getting substitution scores for anti-diagonal cells resulted complex and hard to resolve efficiently, because each amino acid pair requires different indexation in substitution matrix. Even so, Wozniak achieved a 2$\times$ acceleration over the fastest serial implementation of the time.

Three years later, in 2000, Rognes and Seeberg [39] introduced a SIMD version using the MMX/SSE extensions, becoming the first to take advantage of these instruction sets. They found that vectorising along the query sequence was faster than vectorising along the anti-diagonals, in spite of having to make more calculations due to ignoring some of the data dependences mentioned in

Subsection 4.1. Another difference with previous solutions lies on Rognes and Seeberg used 8-bit integer data. This fact allowed them to compute a higher number of alignments in parallel at the cost of reducing score representation range to 0-255. This scheme results beneficial because overflow only occurs when sequences are long and/or very similar between them. When overflow occurs, a 16-bit integer version of the algorithm is employed to guarantee correct results. In addition, they introduced the Query Profile technique (QP), which consists in building an auxiliary two-dimensional array of size $|q| \times |\sum|$, where q is the query sequence and $\sum$ is the alphabet. Each row of this array contains the scores of the corresponding query residue against each possible residue in the alphabet. QP technique improves data locality by replacing a random access to the substitution matrix with a linear access to the QP matrix in the algorithm innermost loop. The SIMD instructions usage together with QP technique allowed Rognes and Seeberg to reach a $6\times$ speed-up over an optimised serial implementation.

Later, in 2007, Farrar [9, 10] used SSE2 extensions to develop an improved version of the Rognes and Seeberg implementation. Just as the previous approach, Farrar vectorised along the query sequence. However, the alignment matrix computations were re-organised to do them in a striped manner. This access pattern minimises data dependences impact and reduces misaligned vector accesses. Additionally, Farrar proposed the *lazy F* technique, which helps to minimise the number of conditional jumps inside the innermost loop. As a consequence of these improvements, Farrar reported more than 11 and 20 GCUPS when using four and eight cores, respectively.

A year later, Szalkowski *et al.* [45] proposed some improvements to Farrar approach. They presented a multi-threaded version for both x86 architectures with SSE2 compatibility and Cell/Broadband Engine from IBM. This implementation is known as SWPS3 and reached a peak of 15.7 GCUPS when using a quad-core processor.

Afterwards, in 2011, Rognes [38] presented SWIPE, an implementation for Intel processors with SSSE3 instruction set adopting the inter-task scheme proposed previously by Alpern *et al.* [1]. Rognes also introduced the Score Profile technique (SP) to accelerate the substitution scores extraction. The SP technique is based on constructing an auxiliary $n \times l \times |\sum|$ two-dimensional score array, where n is the length of the database sequence, l is the number of vector lanes and $\sum$ is the alphabet. Since each row of the auxiliary matrix forms an l-lane score vector, its values can be loaded at the same time through a single load vectorial instruction. The main SP disadvantage is that score array must be constructed for each database sequence. Using two six-core processors, SWIPE achieved a peak of 106.2 GCUPS, being up to six times faster than SWPS3 and Farrar approach.

In 2013, Zhao *et al.* [55] presented SSW, a library developed in C/C++ to facilitate SW integration to other genomic applications. SSW adopts Farrar approach to compute optimal alignment as well as optimal score. As SSW is also available as an autonomous tool, the authors measured its performance while searching Swiss-Prot database; SSW reported up to 2.53 GCUPS using an AMD x86 64 2.0Ghz processor.

Two years later, in 2015, Rucci *et al.* [41] introduced SWIMM, an implementation for Intel heterogeneous systems combining Xeon and Xeon Phi processors. SWIMM adopts the inter-task scheme and offers three execution mode: (1) Xeon, (2) Xeon Phi and (3) concurrent Xeon and Xeon Phi. Unlike previous implementations, SWIMM is able to take advantage of AVX2 as well as SSE extensions. The AVX2 exploitation allows SWIMM to compute up to 32 alignments in parallel in place of 16 (SSE case). In their work, the authors showed that SWIMM can be comparable with SWIPE when using SSE instruction set. However, when taking advantage of AVX2 extensions, SWIMM demonstrated to be superior to SWIPE. In the Xeon mode with AVX2 exploitation, SWIMM reached up to 360 GCUPS using two Intel Xeon E5-2695 v3 2.3 GHz processors when searching Environmental NR database.

Finally, in 2016, Daily [7] presented Parasail, a C-based library containing implementations of different pairwise sequence alignment algorithms. The intent of this library is to be integrated into other software packages, not necessarily to replace already highly performing database search tools. In that sense, Parasail implements most known algorithms for vectorised SW sequence alignment that follows intra-task parallelism scheme (including Wozniak [52], Rognes and Seeberg [39] and Farrar [9, 10] approaches described above). Additionally, the Parasail library implements each of these methods for different instructions sets: SSE2, SSE4.1, AVX2, and KNC. In his work, the author showed that Parasail's AVX2 implementation is able to outperform SWIPE for query sequences longer than approximately 500 amino acids. Parasail reported up to 291.5 GCUPS using two Intel Xeon E5-2670 2.3 GHz processors when searching UniProt Knowledgebase database.

Table 1 summarises the performance of CPU implementations.

4.2.2 GPU Implementations

The first GPU implementations date back to 2006 and they were proposed by Weiguo Liu *et al.* [22] and Yang Liu *et al.* [23]. Both proposals are very similar: they are based on the OpenGL library, compute alignment matrices by anti-diagonals and store sequences as well as auxiliary buffers in texture memory. Weiguo Liu *et al.* implementation only processes protein sequences of length shorter than 4096 amino acids due to limitations imposed by the texture memory of that time, and for that reason, the experimental work was carried out with a reduced version of Swiss-Prot database (99.8% of original sequences). This implementation reached 0.65 GCUPS using a NVIDIA GeForce 6800 GT, which represented a speed-up of 16$\times$ over a serial CPU implementation. On its behalf, the Yang Liu *et al.* implementation does not impose restrictions on the sequence length and it offers two execution modes: (1) optimal alignment and (2) optimal alignment score. Using a NVIDIA GeForce 7800 GTX, this implementation achieved 0.18 and 0.24 GCUPS in (1) and (2) execution modes, respectively.

Table 1 Performance summary of CPU implementations

Year	Implementation	Hardware used	No. of threads	GCUPS
1997	Wozniak [52]	Sun Ultra Sparc Enterprise 6000 167 MHz	12	0.2
2000	Rognes and Seeberg [39]	Intel Pentium III 500 MHz	1	0.15
2007	Farrar [9]	Intel Xeon Core 2 Duo 2.0 GHz	1	2.9
2008	SWPS3, Szalkowski *et al.* [45]	Intel Core 2 Quad Q6600 2.4 GHz	4	15.07
2011	SWIPE, Rognes [38]	2×Intel Xeon X5650 2.67 GHz	24	106.2
2013	SSW, Zhao *et al.* [55]	AMD x86 64 2.0Ghz	1	2.53
2015	SWIMM, Rucci *et al.* [41]	2×Intel Xeon E5-2695 v3 2.3 GHz	28	309.3
			56	360
2016	Parasail, Jeff Daily [7]	2×Intel Xeon E5-2670 2.3 GHz	24	291.5

In 2008, Manavski and Valle [28] presented the first CUDA implementation for SW protein database search, which was named SW-CUDA. As opposed to previous proposals, SW-CUDA adopts inter-task parallelism scheme because each CUDA thread computes a complete alignment between the query sequence and a particular database sequence. Another distinctive characteristic of SW-CUDA is the QP technique usage to obtain scores from substitution matrix. SW-CUDA reported 1.85 GCUPS using a NVIDIA GeForce 8800 GTX card when searching Swiss-Prot database. Additionally, it showed good scalability with the amount of GPUs owing to a dynamic workload balance technique that considers the compute power of each particular device.

In 2009, Yongchao Liu *et al.* [24] introduced CUDASW++, an implementation for CUDA-enabled GPUs that combines both intra-task and inter-task parallelism approaches. The inter-task scheme usually reports better performance than intra-task scheme: however, it requires more memory resources. Therefore, CUDASW++ sets a configurable length threshold to compute the alignments. Database sequences of length less than or equal to the threshold are computed according to the inter-task scheme. The alignments of database sequences of length greater than the threshold are carried out following the intra-task scheme. CUDASW++ also carefully arranges memory accesses to get coalesced access. Besides, it exploits memory hierarchy by storing query sequence and substitution matrix in constant and shared memories, respectively. This set of optimisations allowed CUDASW++ to reach 9.63 and 16.09 when searching Swiss-Prot database using a single-GPU NVIDIA GeForce GTX 280 and a dual-GPU NVIDIA GeForce GTX 295, respectively.

A year later, the same authors of CUDASW++ presented an improved version of this tool, known as CUDASW++ 2.0 [26]. This version offers two execution

modes: the first one adopts the original mode but includes QP technique and replaces scalar data with packed data; the second one follows the Farrar approach [9] through SIMD instruction virtualization on graphic cards. Searching Swiss-Prot database, CUDASW++ 2.0 reported similar behaviour between both execution modes, reaching 16.9 and 29.6 GCUPS on a single-GPU NVIDIA GeForce GTX 280 and on a dual-GPU NVIDIA GeForce GTX 295, respectively.

GASW is another tool for CUDA-compatible GPUs [19]. This software was introduced in 2010 and among its optimisation we can mention elimination of memory bottlenecks and the conversion of the database to a format convenient for GPU usage. GASW achieved up to 21.36 GCUPS on a NVIDIA GeForce GTX 275 when searching Swiss-Prot database.

In 2011, Zou *et al.* [56] presented a CUDA-based implementation that combines different optimisations: global memory accesses in coalescent manner, hierarchy memory exploitation and loop unrolling. This implementation reached 28.35 GCUPS on a NVIDIA GeForce GTX 470 when searching a synthetic database of 107520 sequences (each sequence contains 1024 random residues).

There are few known implementations based on OpenCL for GPUs. Khalafallah *et al.* [20] follows inter-task approach and reuses several optimisations from previous implementations like global memory accesses arrangement to obtain coalescent access and texture memory usage to store QP matrix. This implementation achieved 12.29 and 65.99 GCUPS when searching a reduced version of Swiss-Prot database on a NVIDIA GeForce 9800 GT and an ATI HD 5850, respectively. On its behalf, Borovska and Lazarova proposal [6] also adopts inter-task scheme, although it does not provide enough implementation details. With Swiss-Prot database, this software reported 1.6 and 7.8 GCUPS on a NVIDIA Quadro FX3600M and on a NVIDIA GeForce GTX 295 (dual-GPU), respectively.

Finally, in 2013, Yongchao Liu *et al.* [27] presented a third version of CUDASW++, which was named CUDASW++ 3.0. This implementation targets NVIDIA GPUs based on the Kepler architecture and combines concurrent CPU and GPU computations adopting the inter-task approach on both cases. The workload is distributed dynamically using an heuristic based on hardware characteristics and constants derived from empirical evaluations. For the GPU computation, CUDASW++ 3.0 employs CUDA PTX video instructions. For the CPU computation, this algorithm uses SSE extensions; specially the host code is based on SWIPE code. Alignments are computed using 8-bit integers on both CPU and GPU devices. Once all alignments are processed, CPU detects overflow cases and recomputes them using 16-bit integer. It is important to mention that, because GPU adopts inter-task scheme, it only processes those database sequences of length less than, or equal to, the threshold. Longer sequences are obligatory computed in CPU. With Swiss-Prot as benchmark database, CUDASW++ 3.0 reached 119 GCUPS using a personal computer based on a quad-core Intel i7 2700k 3.5Ghz processor and a NVIDIA GeForce GTX 680. Because NVIDIA decided to cut down the capability of the PTX video instructions in Maxwell architecture, CUDASW++ 3.0 could not run at full speed on these GPUs. Beyond the previous

limitation, CUDASW++ 3.0 is considered the fastest SW implementation for CUDA-compatible systems as of today.

Table 2 summarises the performance of GPU implementations.

4.2.3 FPGA Implementations

Acceleration of sequence alignment using FPGAs is a widely studied topic in HPC community. However, most of these implementations focus on DNA alignment because it is simpler than protein alignment from an algorithmic perspective (DNA alignment has a reduced alphabet and adopts a simpler scoring scheme).

Beyond sequence type, FPGA implementations are usually based on creating basic building blocks that can compute a matrix cell in a clock cycle. Next, multiple instances of these blocks are combined at the same time to create systolic arrays capable of processing large amounts of data in parallel. A systolic array is an arrangement of processing units in array form, where data flows synchronously among the units, usually in a specific direction. This kind of array works like vectorial units in modern CPUs (e.g. SSE units) but, instead of having a fixed length, systolic arrays can configure its length [48].

Unfortunately, analytic comparison among these implementations is quite difficult due to different causes [8, 14]:

- There is a wide variety of FPGAs and each of them implements its circuitry in a different way, which complicates direct comparison.
- There are very few fully functional tools. The implementations that report the best performance only contemplate synthetic tests with the aim to show the potential of this kind of accelerator although its usage in real world is very limited. Some implementations effectively employ real data but also present some limitations: query sequence is embedded in the design, sequences have a fixed or limited length, search parameters (gap penalizations or substitution matrix) are fixed or they require hardware reconfiguration to change them, among others.
- Lack of documentation on implementation details. For example, the implementation performance depends strongly on the data width used. Normally, performance improves as data width reduces, although a very small size could be insufficient to compute all alignments. Researchers tend to omit this kind of detail.

Among the fully functional implementations, we can found Isa *et al.* [18], Benkrid *et al.* [4] and Rucci *et al.* [40]. In 2011, Isa *et al.* [18] proposed a linear systolic array implementation for a Xilinx Virtex-5 XC5VLX110 FPGA although programming language was not specified. This implementation consists of a pipeline of basic processing elements, each holding one query residue whereas the subject sequence is shifted systolically through it. The alignments are computed using 11-bit integers. When searching Swiss-Prot database, this implementation reached up to 28 GCUPS.

Table 2 Performance summary of GPU implementations

Year	Implementation	Programming model	Hardware used	Database	GCUPS
2006	Weiguo Liu *et al.* [22]	OpenGL	NVIDIA GeForce 6800 GT	Reduced Swiss-Prot (99.8%)	0.65
2006	Yang Liu *et al.* [23]	OpenGL	NVIDIA GeForce 7800 GTX	983 sequences (462862 amino acids)	0.24
2008	SW-CUDA, Manavski and Valle [28]	CUDA	NVIDIA GeForce 8800 GTX	Swiss-Prot	1.85
			2×NVIDIA GeForce 8800 GTX		3.61
2009	CUDASW++, Yongchao Liu *et al.* [24]	CUDA	NVIDIA GeForce GTX 280	Swiss-Prot	9.63
			NVIDIA GeForce GTX 295 (dual-GPU)		16.09
2010	CUDASW++ 2.0, Yongchao Liu *et al.* [26]	CUDA	NVIDIA GeForce GTX 280	Swiss-Prot	16.9
			NVIDIA GeForce GTX 295 (dual-GPU)		29.6
2010	GASW, Kentie [19]	CUDA	NVIDIA GeForce GTX 275	Swiss-Prot	21.36
2010	Khalafallah *et al.* [20]	OpenCL	NVIDIA GeForce 9800 GT	Reduced Swiss-Prot (45%)	12.29
			ATI HD 5850		65.99
2011	Zou *et al.* [56]	CUDA	NVIDIA GeForce GTX 280	107520 sequences of 1024 amino acids each	13.71
			NVIDIA GeForce GTX 470		28.35
2011	Borovska and Lazarova [6]	OpenCL	NVIDIA Quadro FX3600M	Swiss-Prot	1.6
			NVIDIA GeForce GTX 295 (dual-GPU)		7.8
2013	CUDASW++ 3.0, Yongchao Liu *et al.* [27]	Pthreads + CUDA	Intel i7 2700k 3.5Ghz + NVIDIA GeForce GTX 680	Swiss-Prot	119
			Intel i7 2700k 3.5Ghz + NVIDIA GeForce GTX 690 (dual-GPU)		185.6

A year later, Benkrid *et al.* [4] introduced an implementation similar to Isa *et al.* proposal [18]. This implementation is also based on a linear systolic array. However, alignments are computed using 16-bit integers and the corresponding FPGA design was captured in a C-based high level hardware language, called Handel-C [30]. Using a Xilinx Virtex-4 LX160-11 FPGA, the authors reported up to 19.4 GCUPS when searching Swiss-Prot database.

In 2016, Rucci *et al.* [40] presented OSWALD, a tool developed with Altera OpenCL SDK for Altera FPGA-based systems. Unlike the rest of the implementations, this tool does not follow a linear systolic array fashion, on the contrary, it adopts the inter-task scheme. OSWALD computes alignments in FPGA using 8-bit integers and the host recomputes overflowed alignments using wider integer data. Also, it is able to combine concurrent CPU computations through multi-threading and SIMD exploitation. On a heterogeneous platform based on two Xeon E5-2670 and a single Altera Stratix V GSD5 FPGA, OSWALD reached up to 58.4 GCUPS on FPGA mode and 178.9 GCUPS on hybrid mode (host+FPGA), while searching Environmental NR database.

Table 3 summarises the performance of known FPGA implementations.

4.2.4 Xeon Phi Implementations

Xeon Phi coprocessors can also be employed to accelerate SW alignments. In 2014, Liu and Schmidt [25] introduced SWAPHI, a tool for similarity searches based on OpenMP. This implementation adopts the offload model and is capable of taking advantage of several coprocessors at the same time. In this work, the authors explored the benefits of intra-task and inter-task approaches as well as QP and SP techniques. In particular, SWAPHI is able to compute 16 cells in parallel due to KNC instruction set. Using a Xeon Phi 5110P, SWAPHI achieved up to 45.6 and 58.8 when searching TrEMBL database with intra-task and inter-task schemes, respectively. When using four coprocessors, performance increased to 164.9 (intra-task) and 228.4 GCUPS (inter-task) for the same database searches.

XSW is another similarity search tool based on Xeon Phi coprocessors [50]. Like SWAPHI, XSW employs inter-task scheme, SP technique and the KNC instruction set. Unlike SWAPHI, XSW works in native mode. XSW reported up to 70 GCUPS when searching Environmental NR database using a Xeon Phi 7110P.

An extended version of XSW, known as XSW 2.0, was developed subsequently by the same authors [51]. This implementation follows the offload model and combines concurrent CPU computations through multi-threading and SSE extensions. Using a Intel Xeon E5-2620 processor and a Xeon Phi 7110P coprocessor, XSW 2.0 reached up to 100 GCUPS when searching Environmental NR database.

In 2015, Rucci *et al.* [41] presented the previously described SWIMM tool. In their work, the authors state that, despite having more cores and wider vectorial processing units, the poor performance (in terms of GCUPS) of the Xeon Phi compared to Xeon is due mainly to the absence of low-range vector capabilities

Table 3 Performance summary of FPGA implementations

Year	Implementation	Programming language	Hardware used	GCUPS	Comments
2004	Dydel and Bala [8]	VHDL	Xilinx Virtex II XC2VP70-5	11.18	Synthetic tests. Fixed sequence lengths.
2005	Oliver *et al.* [35]	Verilog	Xilinx Virtex II XC2V6000	10.6	Query sequence of limited length. Require hardware reconfiguration to change search parameters.
2007	Zhang *et al.* [54]	Not specified	Altera Stratix II EP25180	25.6	Require hardware reconfiguration to change search parameters
2007	Van Court and Herbordt [47]	VHDL	Xilinx Virtex II Pro XC2VP70-5	5.41	Sequences of limited length. Require hardware reconfiguration to change search parameters.
2009	Benkrid *et al.* [5]	Handel-C	Xilinx Virtex II XC2V6000-4	7.66*	Require hardware reconfiguration to change search parameters. *Theorethical peak performance.
2011	Isa *et al.* [18]	Not specified	Xilinx Virtex-5 XC5VLX110	28	
2011	Zou *et al.* [56]	Not specified	Xilinx Virtex-5 XC5VLX330	47	Synthetic tests
2012	Benkrid *et al.* [4]	Handel-C	Xilinx Virtex-4 LX160-11	19.4	
2016	OSWALD, Rucci *et al.* [40]	OpenMP + OpenCL	2×Intel Xeon E5-2670 2.60Ghz + Altera Stratix V GSD5	58.4 178.9	FPGA mode Hybrid mode

on the Xeon Phi. Beyond that, SWIMM showed to be comparable with SWAPHI in the Xeon Phi mode and significantly superior to XSW 2.0 in the hybrid mode. When searching Environmental NR database, SWIMM reported 160 GCUPS using two Intel Xeon E5-2670 2.60Ghz processors and an Intel Xeon Phi 3120P.

Finally, XSW 2.0 was replaced by another tool named LSBDS [21]. The main difference between XSW 2.0 and LSBDS is that the latter adopted a multi-pass method to compute the alignment matrices that solved the performance drop problem for long query sequences of the former. LSBDS reported up to 220 GCUPS when searching a merged database (Environmental NR + TrEMBL) using two Intel Xeon E5-2620 v2 2.0Ghz processors and two Intel Xeon Phi 7110P.

Table 4 summarises the performance of Xeon Phi implementations.

Table 4 Performance summary of Xeon Phi implementations

Year	Implementation	Programming model	Hardware used	Database	GCUPS
2014	SWAPHI, Liu and Schmidt [25]	OpenMP (offload)	Intel Xeon Phi 5110P	TrEMBL	58.8
			4× Intel Xeon Phi 5110P		228.4
2014	XSW, Wang *et al.* [50]	OpenMP + Pthreads (native)	Intel Xeon Phi 7110P	Environmental NR	70
2014	XSW 2.0, Wang *et al.* [51]	OpenMP + Pthreads (offload)	Intel Xeon E5-2620 + Intel Xeon Phi 7110P	Environmental NR	100
2015	SWIMM, Rucci *et al.* [41]	OpenMP (offload)	2×Intel Xeon E5-2670 2.60Ghz + Intel Xeon Phi 3120P	Environmental NR	160
2015	LSBDS, Lan *et al.* [21]	OpenMP + Pthreads (offload)	2×Intel Xeon E5-2620 v2 2.0Ghz + 2×Intel Xeon Phi 7110P	TrEMBL + Environmental NR	220

5 Performance/Power Consumption Evaluations

Energy efficiency is becoming more important every day in the HPC community. There is a wide availability of works exploring performance and power consumption of different hardware devices. However, only three do it in the SW context. All of these works have similar coarse-grain results but differ in the improvement coefficients due to different methodological aspects.

In 2011, Zhou *et al.* [56] evaluated performance and power consumption of several implementations for CPU, GPU and FPGA. Considering energy efficiency as GCUPS/Watt, they found that FPGA outperforms CPU and GPU by factors of 50× and 26×, respectively. These significant differences can be explained according to three reasons. In the first place, host consumption was not considered in GPU and FPGA implementations to the detriment of CPU implementations. In the second place, CPU and GPU implementations are sub-optimal. These implementations reported less GCUPS than others previously presented in literature. In the third place, they employed synthetic data benefiting FPGA implementations, besides being non-representative of real world biological searches.

Benkrid *et al.* [4] is another performance/power consumption evaluation in the SW context. In this work, the authors presented some implementations for different hardware devices: CPU, GPU and FPGA. They found that FPGA is the most energy efficient platform being up to 500× and 23× better than CPU and GPU, respectively. These impressive results are explained due to several reasons. Firstly, host consumption was not considered in FPGA and GPU cases as in the previous work. Secondly, the authors state that they chose specific hardware platforms

based on the same fabrication technology (90nm) to enable a fair comparison between devices. This fact clearly benefits FPGA and GPU implementations since CPUs are more advanced in this aspect. Lastly, they employed sub-optimal implementations for all devices. As the work also evaluates programming cost of each implementation, they chose not to use the results of the fastest implementations reported in the literature, but instead to perform their own experiments using solutions developed by a set of Ph.D. students with relatively equal experience on each platform.

Rucci *et al.* [40] is probably the most realistic SW performance/power consumption evaluation, since the authors considered host consumption in accelerator versions and employed powerful hardware platforms, the best implementations available in literature so far for each device and real biological data. Taking CPU-based systems as baseline, these authors compared performance and energy efficiency of hybrid systems using all its available computational resources (host and accelerators). They found that CPUs offer a good balance between performance and power consumption, especially those with AVX2 instruction set. Xeon Phi-based systems are not a good choice for this problem from energy efficiency perspective principally due to the absence of low-range vector capabilities on this coprocessor. The performance gain is smaller than the increase in power consumption, which translates into less GCUPS/Watt. Both CPU-FPGA and CPU-GPU systems are able to improve energy efficiency, being the first step forward to the second. GPU accelerated systems offer higher performance rates but at the expense of higher power consumption rates too. CPU-FPGA systems offer less GCUPS than GPU-based platforms. However, because its power consumptions is lower, the energy efficiency rates are higher. In particular, GPU incorporation performed up to 1.6× and 1.22× in performance and energy efficiency points of view, respectively. In the FPGA case, its addition produced improvements of up to 1.4× in performance and 1.28× in energy efficiency.

6 Future View

Biological data continue increasing its size and accelerating SW database searches still remains as a challenging task. Fortunately, several new hardware and software technologies will be upcoming in the near future that will help to mitigate this issue:

- Multi-core processors offer a good balance between performance and energy efficiency for SW database searches, in particular those with AVX2 instruction set. Future processors of this kind will have more cores, extended vector units and more complex and larger memory hierarchies. As explained in Section 4.2.1, CPU-based implementations can be benefited directly from these features.
- GPUs have demonstrated to be powerful platforms to accelerate SW algorithm. Next generation GPUs will have more computational power and better memory performance. These characteristics can lead to faster implementations.

- FPGAs have also proved to be a good option for accelerating SW protein searches, specially from energy efficiency perspective. In this way, new hybrid CPU-FPGA architectures appear as a promising opportunity.
- Current generation of Xeon Phi are not a good alternative for accelerating SW protein searches due mainly to the absence of low-range vector capabilities on this coprocessor. Fortunately, next generation of Xeon Phi (Knights Landing) will solve this issue incorporating the AVX-512 instruction set. Therefore, better Xeon Phi performances are expected considering the number of cores and the vector capabilities.

Some of these technologies will deliver improvements in a transparent manner to programmers. For example, all hardware vendors plan to reduce the manufacturing process technology of their devices, which can turn into faster communications and more available computational resources. Or also stacked memory adoption that will increase bandwidth and energetic efficiency. However, other technologies will require programmer's intervention to take advantage of them, like new hybrid CPU-FPGA architectures or also the AVX-512 instruction set that will be available in Xeon Skylake processors and next generation Xeon Phi coprocessors. Therefore, programming efforts will be necessary to develop new computational tools capable of taking advantage of these upcoming technologies.

7 Conclusions

In this chapter we gave a survey of the state-of-the-art in SW database protein search, focusing on four widespread hardware architectures: CPU, GPU, FPGA and Xeon Phi. First, we presented a brief description of each platform. Next, we explained the SW algorithm followed by the study of its data dependences and the possible parallelism schemes. We reviewed the existing implementations for the hardware platforms under study including temporal evolution, contributions, limitations and experimental work and results of each of them. Additionally, as energy efficiency is becoming more important every day, we also surveyed performance/power consumption works in SW context. Finally, we gave our view on the future of SW protein searches considering next generations of hardware architectures and its upcoming technologies.

Biological data continue increasing its size and, as a consequence, increasing SW search time. Upcoming technologies will help to mitigate this issue but will also present new challenges to programmers in order to take advantage of them. We expect that this chapter can serve as a good starting point to future acceleration of SW protein database searches.

Acknowledgements Enzo Rucci holds a PhD CONICET Fellowship from the Argentinian Government, and this work has been partially supported by Spanish research project TIN 2012-32180.

References

1. Alpern B, Carter L and Gatlin KS (1995) Microparallelism and High-performance Protein Matching. SC95, doi:10.1109/SUPERC.1995.242795
2. Altera Corporation (2016) Altera SDK for OpenCL. Available at https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html Cited 08 Jan 2016
3. AMD (2016) High-Bandwidth Memory. Available at http://www.amd.com/en-us/innovations/software-technologies/hbm Cited 08 Jan 2016
4. Benkrid K, Akoglu A, Ling C, Song Y, Liu Y and Tian X (2012) High performance biological pairwise sequence alignment: FPGA versus GPU versus cell BE versus GPP. Int. J. Reconfig. Comput., doi:10.1155/2012/752910
5. Benkrid K, Ying L and Benkrid A (2009) A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, doi:10.1109/TVLSI.2008.2005314
6. Borovska P and Lazarova M (2011) Parallel models for sequence alignment on CPU and GPU. CompSysTech 2011, doi:10.1145/2023607.2023644
7. Daily J (2016) Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. BMC Bioinformatics, doi: 10.1186/s12859-016-0930-z
8. Dydel S and Bala P (2004) Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices. LNCS, doi:10.1007/978-3-540-30117-2_ 5
9. Farrar M (2007) Striped Smith-Waterman speeds database searches six time over other SIMD implementations. Bioinformatics, doi:10.1093/bioinformatics/btl582
10. Farrar M (2008) Optimizing Smith-Waterman for the Cell Broad-band Engine. Available at http://farrar.michael.googlepages.com/SW-CellBE.pdf Cited 21 Mar 2009
11. Giles MB and Reguly I (2014) Trends in high-performance computing for engineering calculations. Philos Trans A Math Phys Eng Sci., doi:10.1098/rsta.2013.0319
12. Gotoh O (1982) An improved algorithm for matching biological sequences. J. Mol. Biol., doi:10.1016/0022-2836(82)90398-9
13. Harris M (2014) Maxwell: The Most Advanced CUDA GPU Ever Made. Available at http://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/ Cited 08 Jan 2016
14. Hasan L and Al-Ars Z (2011) An Overview of Hardware-Based Acceleration of Biological Sequence Alignment. In: Lopes H (ed) Computational Biology and Applied Bioinformatics. InTech
15. Howse B and Smith R (2015) Tick Tock On The Rocks: Intel Delays 10nm, Adds 3rd Gen 14nm Core Product Kaby Lake. Available at http://www.anandtech.com/show/9447/intel-10nm-and-kaby-lake Cited 08 Dec 2015
16. IBM (2015) IBM and Xilinx Announce Strategic Collaboration to Accelerate Data Center Applications. Available at https://www-03.ibm.com/press/us/en/pressrelease/48074.wss Cited 18 Jan 2016
17. Intel (2016) Intel Acquisition of Altera. Available at intelacquiresaltera.transactionannouncement.com Cited 18 Jan 2016
18. Isa MN, Benkrid K, Clayton T, Ling C and Erdogan AT (2011) An FPGA-based parameterised and scalable optimal solutions for pairwise biological sequence analysis. AHS 2011, doi:10.1109/AHS.2011.5963957
19. Kentie M (2010) Biological Sequence Alignment Using Graphics Processing Units. MSc Thesis, TUDelft
20. Khalafallah A, Elbabb HF, Mahmoud O and Elshamy A (2010) Optimizing Smith-Waterman algorithm on Graphics Processing Unit. ICCTD 2010, doi:10.1109/ICCTD.2010.5645976
21. Lan h, Liu W, Schmidt B, and Wang B (2015) Accelerating Large-Scale Biological Database Search on Xeon Phi-based Neo-Heterogeneous Architectures. BIBM 2015, doi:10.1109/BIBM.2015.7359735

22. Liu W, Schmidt B, Voss G, Schroder A and Muller-Wittig W (2006) Bio-sequence database scanning on a GPU. IPDPS 2006, doi:IPDPS.2006.1639531
23. Liu Y, Huang W, Johnson J and Vaidya S (2006) GPU Accelerated Smith-Waterman. LNCS, doi:10.1007/11758549_29
24. Liu Y, Maskell DL and Schmidt B (2009) CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. BMC Research Notes, doi:10.1186/1756-0500-2-73
25. Liu Y and Schmidt B (2014) SWAPHI: Smith-Waterman Protein Database Search on Xeon Phi Coprocessors. ASAP 2014, doi:10.1109/ASAP.2014.6868657
26. Liu Y, Schmidt B and Maskell DL (2010) CUDASW++ 2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. BMC Research Notes, doi:10.1186/1756-0500-3-93
27. Liu Y, Wirawan A and Schmidt B (2013) CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. BMC Bioinformatics, doi:10.1186/1471-2105-14-117
28. Manavski S and Valle G (2008) CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. BMC Bioinformatics, doi:10.1186/1471-2105-9-S2-S10
29. McCool MD (2008) Scalable Programming Models for Massively Multicore Processors. Proceedings of the IEEE, doi: 10.1109/JPROC.2008.917731
30. Mentor Graphics (2015) Handel-C System Methodology. Available at `https://www.mentor.com/products/fpga/handel-c/` Cited 08 Jan 2016
31. Moammer K (2015) AMD Zen CPU Microarchitecture Details Leaked In Patch - Doubles Down On IPC And Floating Point Throughput. Available at `http://wccftech.com/amd-zen-cpu-core-microarchitecture-detailed/2/` Cited 16 Oct 2015
32. Moammer K (2015) Nvidia : Pascal Is 10X Maxwell, Launching in 2016 – Features 16nm, 3D Memory, NV-Link and Mixed Precision. Available at `http://wccftech.com/nvidia-pascal-gpu-gtc-2015/` Cited 16 Jan 2016
33. Needleman SB and Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol., doi:10.1016/0022-2836(70)90057-4
34. NVIDIA Corporation (2016) CUDA. Available at `http://www.nvidia.com/object/cuda_home_new.html` Cited 08 Jan 2016
35. Oliver TF, Schmidt B and Maskell DL (2005) Reconfigurable architectures for bio-sequence database scanning on FPGAs. IEEE Transactions on Circuits and Systems, doi:10.1109/TCSII.2005.853340
36. OpenACC Organization (2016) OpenACC. Available at `http://www.openacc.org/` Cited 08 Jan 2016
37. Pirzada U (2015) Intel's Skylake Purley Family of Microprocessors Will Boast upto 28 Cores and 56 Threads - Next Generation Xeon Platform Landing in 2016. Available at `http://wccftech.com/intel-skylake-purley-platform-upto-28-cores-56-threads/` Cited 08 Dec 2015
38. Rognes T (2011) Faster Smith-Waterman database searches with inter-sequence SIMD parallelization. BMC Bioinformatics, doi:10.1186/1471-2105-12-221
39. Rognes T and Seeberg E (2000) Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. Bioinformatics, doi:10.1093/bioinformatics/16.8.699
40. Rucci E (2016) Evaluación de rendimiento y eficiencia energética en sistemas heterogéneos para bioinformática. PhD Thesis, UNLP
41. Rucci E, García C, Botella, G, De Giusti A, Naiouf M and Prieto-Matías M (2015) An energy-aware performance analysis of SWIMM: Smith-Waterman implementation on Intel's Multicore and Manycore architectures. CPE, doi: 10.1002/cpe.3598

42. Seetle S (2013) High-performance Dynamic Programming on FPGAs with OpenCL. Available at http://ieee-hpec.org/2013/index_htm_files/29-High-performance-Settle-2876089.pdf Cited 08 Jan 2016
43. Smith R (2011) AMD's Graphics Core Next Preview: AMD's New GPU, Architected For Compute. Available at http://www.anandtech.com/show/4455/amds-graphics-core-next-preview-amd-architects-for-compute Cited 08 Jan 2016
44. Smith TF and Waterman MS (1981) Identification of common molecular subsequences. J. Mol. Biol., doi:10.1016/0022-2836(81)90087-5
45. Szalkowski A, Ledergerber C, Krahenbuhl P and Dessimoz C (2008) SWPS3 - fast multi-threaded vectorized Smith-Waterman for IBM Cell/B.E. and x86/SSE2. BMC Research Notes, doi:10.1186/1756-0500-1-107
46. The Khronos Group (2016) OpenCL: The open standard for parallel programming of heterogeneous systems. Available at https://www.khronos.org/opencl/ Cited 08 Jan 2016
47. Van Court T and Herbordt MC (2004) Families of FPGA-based algorithms for approximate string matching. ASAP 2004, doi:10.1109/ASAP.2004.1342484
48. Vermij E (2011) Genetic sequence alignment on a supercomputing platform. MSc Thesis, TUDelft
49. Vestias M and Neto H (2014) Trends of CPU, GPU and FPGA for high-performance computing. FPL 2014, doi:10.1109/FPL.2014.6927483
50. Wang L, Chan Y, Duan X, Lan H, Meng X and Liu W (2014) XSW: Accelerating Biological Database Search on Xeon Phi. IPDPS 2014, doi:10.1109/IPDPSW.2014.108
51. Wang L, Chan Y, Duan X, Lan H, Meng X and Liu W (2014) XSW 2.0: A fast Smith-Waterman Algorithm Implementation on Intel Xeon Phi Coprocessors. Available at http://sdu-hpcl.github.io/XSW/ Cited 16 Nov 2015
52. Wozniak A (1997) Using video-oriented instructions to speed up sequence comparison. CABIOS 13-2:145–150
53. Xilinx Inc. (2016) SDAccel Development Environment. Available at http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html Cited 08 Jan 2016
54. Zhang P, Tan G and Gao GR (2007) Implementation of the Smith-Waterman Algorithm on a Reconfigurable Supercomputing Platform. HPRCTA 2007, doi:10.1145/1328554.1328565
55. Zhao M, Lee W, Garrison E and Marth G (2013)SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications. PLoS One, doi:10.1371/journal.pone.0082138
56. Zou D, Dou Y and Xia F (2011) Optimization schemes and performance evaluation of Smith-Waterman algorithm on CPU, GPU and FPGA. CPE, doi: 10.1002/cpe.1913